**Day 3 Question: Count Days Without Meetings**

**Problem Statement:**

You are given a positive integer days representing the total number of days an employee is available for work (starting from day 1). You are also given a 2D array meetings of size n, where meetings[i] = [start\_i, end\_i] represents the starting and ending days of meeting i (inclusive).

Return the count of days when the employee is available for work but no meetings are scheduled.

**Note:** The meetings may overlap.

**Example 1:**

**Input:**

days = 10, meetings = [[5,7],[1,3],[9,10]]

**Output:**

2

**Explanation:**  
There is no meeting scheduled on the 4th and 8th days.

**Example 2:**

**Input:**

days = 5, meetings = [[2,4],[1,3]]

**Output:**

1

**Explanation:**  
There is no meeting scheduled on the 5th day.

**Example 3:**

**Input:**

days = 6, meetings = [[1,6]]

**Output:**

0

**Explanation:**  
Meetings are scheduled for all working days.

**Constraints:**

* 1 <= days <= 10^9
* 1 <= meetings.length <= 10^5
* meetings[i].length == 2
* 1 <= meetings[i][0] <= meetings[i][1] <= days

**Optimized Approach:**

1. **Sort Meetings:** Sort the meetings based on the start day.
2. **Merge Overlapping Meetings:** Track the last occupied day to calculate free days efficiently.
3. **Count Available Days:** Subtract the occupied days from the total available days.

**Time Complexity:** O(n log n) (due to sorting) + O(n) (processing) → **O(n log n) overall.**

**Java Solution:**

import java.util.Arrays;

class Solution {

public int countDays(int days, int[][] meetings) {

Arrays.sort(meetings, (a, b) -> Integer.compare(a[0], b[0])); // Sort meetings by start day

int meetingDays = 0, lastEnd = 0;

for (int[] meeting : meetings) {

int start = meeting[0], end = meeting[1];

// Count days without meetings between previous meeting end and current start

if (start > lastEnd + 1) {

meetingDays += (start - lastEnd - 1);

}

// Merge overlapping meetings

lastEnd = Math.max(lastEnd, end);

}

// Count remaining free days after last meeting

if (lastEnd < days) {

meetingDays += (days - lastEnd);

}

return meetingDays;

}

}

**Explanation of Code:**

1. **Sorting Meetings:** Meetings are sorted based on the start day to process them efficiently.
2. **Tracking Free Days:** The algorithm checks gaps between meetings to count available days.
3. **Handling Edge Cases:** If meetings cover all days, the result is 0. If meetings do not start from day 1, it counts initial free days.

**Edge Cases to Consider:**

* When there are no meetings, all days are available.
* Meetings completely cover the range of days.
* Meetings overlap in complex ways (e.g., multiple overlapping intervals).
* Large constraints (up to 10^9 days) make using an array infeasible.

**Final Thoughts:**

* This approach ensures we efficiently count free days without iterating over all days, making it feasible for large inputs.
* Sorting and merging intervals is an optimal way to handle the problem.

🚀 **Happy Coding!**